Prolog Programming Assignment:

Various Computions

Thomas Moskal

CSC 344 – Professor Graci

Due date: 5/9/22

Learning Abstract

This is the final programming assignment of our CSC 344 class. It is the only assignment involving Haskell. Haskell is a general-purpose statically-typed purely functional programming language. In my opinion, it is one of the most intuitive languages that we have learned this semester (Or I've just improved in programming enough that learning it was felt trivial compared to the other languages). In this assignment we evaluate basic syntax of Haskell, the list processing abilities of Haskell as well as higher order functions. Near the end of the assignment, we create a function that does the computations of the "normalized pairwise variability index" or nPVI. And finally, the last task is to play around with a file that can create morse code patterns. We use Haskell to make morse code strings out of the alphabet.

Task 1

Mindfully Mimicked Demonstration

```
>>> length [2,3,5,7]
4
>>> words "need more coffee"
["need","more","coffee"]
>>> unwords ["need","more","coffee"]
"need more coffee"
>>> reverse "need more coffee"
"eeffoc erom deen"
>>> reverse ["need","more","coffee"]
["coffee","more","need"]
>>> head ["need","more","coffee"]
"need"
>>> tail ["need","more","coffee"]
["more","coffee"]
>>> last ["need","more","coffee"]
"coffee"
>>> init ["need","more","coffee"]
["need","more"]
>>> take 7 "need more coffee"
"need mo"
>>> drop 7 "need more coffee"
"re coffee"
>>> ( \x -> length x > 5 ) "Friday"
True
>>> ( \x -> length x > 5 ) "uhoh"
False
>>> ( \x -> x /= ' ' ) 'Q'
True
>>> ( \x -> /= ' ' ) ' '

<interactive>:24:9: error: parse error on input `/='
>>> ( \x -> x /= ' ' ) ' '
False
>>> filter (\x -> x /= ' ' ) "Is the Haskell fun yet?"
"IstheHaskellfunyet?"
>>> :quit
Leaving GHCi.
```

Task 2 – Numeric Function Definitions

Demo

```
[1 of 1] Compiling Main             ( ha.hs, interpreted )
Ok, one module loaded.
>>> blueAreaOfCube 10
482.19027549038276
>>> squareArea 10
100
>>> squareArea 12
144
>>> circleArea 10
314.1592653589793
>>> circleArea 12
452.3893421169302
>>> blueAreaOfCube 10
482.19027549038276
>>> blueAreaOfCube 12
694.3539967061512
>>> blueAreaOfCube 1
4.821902754903828
>>> map blueAreaOfCube [1..3]
[4.821902754903828,19.287611019615312,43.39712479413445]
>>> paintedCube1 1
0
>>> paintedCube1 2
0
>>> paintedCube1 3
6
>>> map paintedCube1 [1..10]
[0,0,6,24,54,96,150,216,294,384]
>>> map paintedCube2 1

<interactive>:24:1: error:
    * Non type-variable argument in the constraint: Num [b]
      (Use FlexibleContexts to permit this)
    * When checking the inferred type
        it :: forall b. (Eq b, Num b, Num [b]) => [b]
>>> paintedCube2 1
0
>>> paintedCube2 2
0
>>> paintedCube2 3
12
>>> map paintedCube2 [1..10]
[0,0,12,24,36,48,60,72,84,96]
```

Code

```haskell
squareArea :: Num a => a -> a
squareArea side = side * side

circleArea :: Floating a => a -> a
circleArea radius = pi * ( radius ^ 2 )

blueAreaOfCube :: Floating a => a -> a
blueAreaOfCube cubeSide = cubeArea - whiteArea where
    cubeArea = 6 * (cubeSide ^ 2)
    whiteArea = 6 * ( circleArea (cubeSide/4) )

paintedCube1 :: (Eq p, Num p) => p -> p
paintedCube1 1 = 0
paintedCube1 2 = 0
paintedCube1 cubeSide = faces * oneFaceBlueCubes where
    oneFaceBlueCubes = ( cubeSide - 2 ) ^ 2
    faces = 6

paintedCube2 :: (Eq p, Num p) => p -> p
paintedCube2 1 = 0
paintedCube2 2 = 0
paintedCube2 cubeSide = numEdges * validEdgeCubes where
    validEdgeCubes = cubeSide - 2
    numEdges = 12

```

Task 3 – Puzzlers

Demo

```
>>> reverseWords "appa and baby yoda are the best"
"best the are yoda baby and appa"
>>> reverseWords "want me some coffee"
"coffee some me want"
>>> reverseWords "reverse in is sentence this"
"this sentence is in reverse"
>>> reverseWords "When all else fails, theres the Uno reverse"
"reverse Uno the theres fails, else all When"
>>> averageWordLength "appa and baby yoda are the best"
3.5714285714285716
>>> averageWordLength "want me some coffee"
4.0
>>> averageWordLength "What is average anyhow?"
5.0
>>> averageWordLength "Oh, it must be five then"
3.1666666666666665
```

Code

```haskell
-------------------------------------------------
--- Task 3

reverseWords :: [Char] -> String
reverseWords str = unwords reversedList where
    reversedList = reverse setOfWords
    setOfWords = words str

averageWordLength :: Fractional a => [Char] -> a
averageWordLength str = fromIntegral amtChar / fromIntegral amtWords where
    setOfWords = words str
    amtWords = length setOfWords
    justChar = filter (/= ' ' ) str
    amtChar = length justChar


```

Task 4 – Recursive List Processors

Demo

```
>>> list2set [1,2,3,2,3,4,3,4,5]
[1,2,3,4,5]
>>> list2set "need more coffee"
"ndmr cofe"
>>> isPalindrome ["coffee","latte","coffee"]
True
>>> isPalindrome ["coffee","latte","espresso","coffee"]
False
>>> isPalindrome [1,2,5,7,11,13,11,7,5,3,2]
False
>>> isPalindrome [2,3,5,7,11,13,11,7,5,3,2]
True
>>> collatz 10
[10,5,16,8,4,2,1]
>>> collatz 11
[11,34,17,52,26,13,40,20,10,5,16,8,4,2,1]
>>> collatz 100\

<interactive>:45:13: error:
    parse error (possibly incorrect indentation or mismatched brackets)
>>> collatz 100
[100,50,25,76,38,19,58,29,88,44,22,11,34,17,52,26,13,40,20,10,5,16,8,4,2,1]
```

Code

```haskell
-----------------------------------------
--- Task 4

list2set :: Eq a => [a] -> [a]
list2set [] = []

list2set (x:xs)
    | x `elem` xs = list2set xs
    | otherwise = x : list2set xs

isPalindrome :: Eq a => [a] -> Bool
isPalindrome [] = True

isPalindrome (x:[]) = True

isPalindrome(x:xs) = response where
    newSet = init xs
    check = x ==  last xs
    response = check && isPalindrome newSet


collatz :: Integral a => a -> [a]
collatz n = cSequence where
    cSequence =
        if n == 1 then [1] else
        if (mod n 2) == 0 then n : (collatz (div n 2)) else
        n : (collatz (3 * n + 1))


```

Task 5 – List Comprehensions

Demo

```
>>> count 'e' "need more coffee"
5
>>> count 4 [1,2,3,2,3,4,3,4,5,4,5,6]
3
>>> count 's' "Gone down Mississippi way to find some solid souls to spend time with"
9
>>> count 2 [1,2,3,2,3,4,3,4,5,4,5,6]
2
>>> freqTable "need more coffee"
[('n',1),('d',1),('m',1),('r',1),(' ',2),('c',1),('o',2),('f',2),('e',5)]
>>> freqTable [1,2,3,2,3,4,3,4,5,4,5,6]
[(1,1),(2,2),(3,3),(4,3),(5,2),(6,1)]
>>> freqTable [1,2,3,2,3,4,3,4,5,4,5,6,5,3,2,1,6,7,8,9,3,2,1,2,3,4,5,6,3,4,3,4,2,4,5,6,200]]

<interactive>:54:88: error: parse error on input `]'
>>> freqTable [1,2,3,2,3,4,3,4,5,4,5,6,5,3,2,1,6,7,8,9,3,2,1,2,3,4,5,6,3,4,3,4,2,4,5,6,200]
[(7,1),(8,1),(9,1),(1,3),(3,8),(2,6),(4,7),(5,5),(6,4),(200,1)]
```

Code

```haskell
72  ----------------------------------------
73  --- Task 5
74
75
76  count :: Eq a => a -> [a] -> Int
77  count obj li = amt where
78      newLi = filter (== obj) li
79      amt = length newLi
80
81  freqTable :: Eq a => [a] -> [(a, Int)]
82  freqTable baseList = finalSet where
83      listSet = list2set baseList
84      amts = map (`count` baseList) listSet
85      finalSet = zipWith (,) listSet amts
86
```

Task 6 – Higher Order Functions

Demo

```
>>> tgl 5
15
>>> tgl 10
55
>>> tgl 15
120
>>> tgl 20
210
>>> triangleSequence 10
[1,3,6,10,15,21,28,36,45,55]
>>> triangleSequence 20
[1,3,6,10,15,21,28,36,45,55,66,78,91,105,120,136,153,171,190,210]
>>> triangleSequence 15
[1,3,6,10,15,21,28,36,45,55,66,78,91,105,120]
>>> triangleSequence 25
[1,3,6,10,15,21,28,36,45,55,66,78,91,105,120,136,153,171,190,210,231,253,276,300,325]
>>> vowelCount "cat"
1
>>> vowelCount "mouse"
3
>>> vowelCount "The cat in the hat chased the mouse"
11
>>> vowelCount "The mouse won"
5
>>> lcsim tgl odd [1..15]
[1,6,15,28,45,66,91,120]
>>> animals = ["elephant","lion","tiger","orangatan","jaguar"]
>>> lcsim length (\w -> elem ( head w ) "aeiou" ) animals
[8,9]
>>> lcsim length (\w -> elem ( head w ) "ltj" ) animals
[4,5,6]
>>> plants = ["lily"

<interactive>:72:17: error:
    parse error (possibly incorrect indentation or mismatched brackets)
>>> plants = ["lily","aloe","coriander","basil","nettles"]
>>> lcsim length (\w -> elem ( head w ) "abc" ) plants
[4,9,5]
```

Code

```haskell
----------------------------------------
--- Task 6

tgl :: (Num b, Enum b) => b -> b
tgl posNum = foldl (+) 0 [1..posNum]

triangleSequence :: (Num b, Enum b) => b -> [b]
triangleSequence posNum = map tgl [1..posNum]

vowelCount :: [Char] -> Int
vowelCount li = numVowels where
    vowelList = filter (\x -> elem x "aeiou") li
    numVowels = length vowelList

lcsim :: (a -> b) -> (a -> Bool) -> [a] -> [b]
lcsim func pred li = map func (filter pred li)

```

Task 7

Demo’s

pairwiseValues

```
>>> pairwiseValues a
[(2,5),(5,1),(1,3)]
>>> pairwiseValues b
[(1,3),(3,6),(6,2),(2,5)]
>>> pairwiseValues c
[(4,4),(4,2),(2,1),(1,1),(1,2),(2,2),(2,4),(4,4),(4,8)]
>>> pairwiseValues u
[(2,2),(2,2),(2,2),(2,2),(2,2),(2,2),(2,2),(2,2),(2,2)]
>>> pairwiseValues x
[(1,9),(9,2),(2,8),(8,3),(3,7),(7,2),(2,8),(8,1),(1,9)]
>>>
```

pairwiseDifferences

```
>>> pairwiseDifferences a
[-3,4,-2]
>>> pairwiseDifferences b
[-2,-3,4,-3]
>>> pairwiseDifferences c
[0,2,1,0,-1,0,-2,0,-4]
>>> pairwiseDifferences u
[0,0,0,0,0,0,0,0,0]
>>> pairwiseDifferences x
[-8,7,-6,5,-4,5,-6,7,-8]
```

pairwiseSums

```
>>> pairwiseSums a
[7,6,4]
>>> pairwiseSums b
[4,9,8,7]
>>> pairwiseSums c
[8,6,3,2,3,4,6,8,12]
>>> pairwiseSums u
[4,4,4,4,4,4,4,4,4]
>>> pairwiseSums x
[10,11,10,11,10,9,10,9,10]
```

pairwiseHalves

```
>>> pairwiseHalves a
[1.0,2.5,0.5,1.5]
>>> pairwiseHalves b
[0.5,1.5,3.0,1.0,2.5]
>>> pairwiseHalves c
[2.0,2.0,1.0,0.5,0.5,1.0,1.0,2.0,2.0,4.0]
>>> pairwiseHalves u
[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0]
>>> pairwiseHalves x
[0.5,4.5,1.0,4.0,1.5,3.5,1.0,4.0,0.5,4.5]
```

pairwiseHalfSums

```
>>> pairwiseHalfSums a
[3.5,3.0,2.0]
>>> pairwiseHalfSums b
[2.0,4.5,4.0,3.5]
>>> pairwiseHalfSums c
[4.0,3.0,1.5,1.0,1.5,2.0,3.0,4.0,6.0]
>>> pairwiseHalfSums u
[2.0,2.0,2.0,2.0,2.0,2.0,2.0,2.0,2.0]
>>> pairwiseHalfSums x
[5.0,5.5,5.0,5.5,5.0,4.5,5.0,4.5,5.0]
```

pairwiseTermPair

```
>>> pairwiseTermPairs a
[(-3,3.5),(4,3.0),(-2,2.0)]
>>> pairwiseTermPairs b
[(-2,2.0),(-3,4.5),(4,4.0),(-3,3.5)]
>>> pairwiseTermPairs c
[(0,4.0),(2,3.0),(1,1.5),(0,1.0),(-1,1.5),(0,2.0),(-2,3.0),(0,4.0),(-4,6.0)]
>>> pairwiseTermPairs u
[(0,2.0),(0,2.0),(0,2.0),(0,2.0),(0,2.0),(0,2.0),(0,2.0),(0,2.0),(0,2.0)]
>>> pairwiseTermPairs x
[(-8,5.0),(7,5.5),(-6,5.0),(5,5.5),(-4,5.0),(5,4.5),(-6,5.0),(7,4.5),(-8,5.0)]
```

pairwiseTerms

```
>>> pairwiseTerms a
[0.8571428571428571,1.3333333333333333,1.0]
>>> pairwiseTerms b
[1.0,0.6666666666666666,1.0,0.8571428571428571]
>>> pairwiseTerms c
[0.0,0.6666666666666666,0.6666666666666666,0.0,0.6666666666666666,0.0,0.6666666666666666,0.0,0.6666666666666666]
>>> pairwiseTerms u
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0]
>>> pairwiseTerms x
[1.6,1.2727272727272727,1.2,0.9090909090909091,0.8,1.1111111111111112,1.2,1.5555555555555556,1.6]
```

nPVI

```
>>> nPVI a
106.34920634920636
>>> nPVI b
88.09523809523809
>>> nPVI c
37.03703703703703
>>> nPVI u
0.0
>>> nPVI x
124.98316498316497
```

Code

```haskell
--- This file is for the creation of a function
--- The function will implement the
--- "normalized pairwise variability index"

a :: [Int]
a = [2,5,1,3]

b :: [Int]
b = [1,3,6,2,5]

c :: [Int]
c = [4,4,2,1,1,2,2,4,4,8]

u :: [Int]
u = [2,2,2,2,2,2,2,2,2,2]

x :: [Int]
x = [1,9,2,8,3,7,2,8,1,9]

pairwiseValues :: [b] -> [(b, b)] | pairwiseValues :: [b] -> [(b, b)] | pairwiseValues :: [b] -> [(b, b)]
pairwiseValues intList = zip (init intList) (tail intList)

pairwiseDifferences :: Num b => [b] -> [b] | pairwiseDifferences :: Num b => [b] -> [b] | pairwiseDifferences :: Num b => [b] -> [b]
pairwiseDifferences intList = map (\(x,y) -> x - y) (pairwiseValues intList)

---pairwiseDifferences intList = zipWith (-) (init intList) (tail intList)

pairwiseSums :: [Int] -> [Int]
pairwiseSums intList = map (\(x,y) -> x + y) (pairwiseValues intList)

half :: Int -> Double
half number = ( fromIntegral number ) / 2

pairwiseHalves :: [Int] -> [Double]
pairwiseHalves intList = map half intList

pairwiseHalfSums :: [Int] -> [Double]
pairwiseHalfSums intList = pairwiseHalves (pairwiseSums intList)

pairwiseTermPairs :: [Int] -> [(Int, Double)]
pairwiseTermPairs intList = zip (pairwiseDifferences intList) (pairwiseHalfSums intList)

term :: (Int,Double) -> Double
term ndPair = abs ( fromIntegral ( fst ndPair ) / ( snd ndPair ) )

pairwiseTerms :: [Int] -> [Double]
pairwiseTerms intList = map term (pairwiseTermPairs intList)

nPVI :: [Int] -> Double
nPVI xs = normalizer xs * sum (pairwiseTerms xs)
   where normalizer xs = 100 / fromIntegral ( ( length xs ) - 1 )
```

Task 8

Subtask a & b

```
>>> dit
"-"
>>> dah
"---"
>>> dit+++dah
"- ---"
>>> m
('m',"--- ---")
>>> g
('g',"--- --- -")
>>> h
('h',"- - - -")
>>> symbols
[('a',"- ---"),('b',"--- - - -"),('c',"--- - --- -"),('d',"--- - -"),('e',"-"),('f',"- - --- -"),('g',"--- --- -"),('h',"- - - -"),('i',"- -"),('j',"- --- --- ---")
,('k',"--- - ---"),('l',"- --- - -"),('m',"--- ---"),('n',"--- -"),('o',"--- --- ---"),('p',"- --- --- -"),('q',"--- --- - ---"),('r',"- --- -"),('s',"- - -"),('t',
"---"),('u',"- - ---"),('v',"- - - ---"),('w',"- --- ---"),('x',"--- - - ---"),('y',"--- - --- ---"),('z',"--- --- - -")]
>>> assoc 't' symbols
('t',"---")
>>> assoc 'o' symbols
('o',"--- --- ---")
>>> assoc 'm' symbols
('m',"--- ---")
>>> find 'a'
"- ---"
>>> find 'n'
"--- -"
>>> fidn 'd'

<interactive>:59:1: error:
    * Variable not in scope: fidn :: Char -> t
    * Perhaps you meant `find' (line 72)
>>> find 'd'
"--- - -"
```

Subtask c & d

```
>>> droplast3 odiumReigns
"--- --- ---   --- - -   - -   - - ---   --- ---      - --- -   -   - -   --- --- -   --- -   - "
>>> droplast7 odiumReigns
"--- --- ---   --- - -   - -   - - ---   --- ---      - --- -   -   - -   --- --- -   --- - "
>>> encodeletter 'm'
"--- ---"
>>> encodeletter 'i'
"- -"
>>> encodeletter 't'
"---"
>>> encodeletter 'yay'

<interactive>:77:14: error:
    * Syntax error on 'yay'
      Perhaps you intended to use TemplateHaskell or TemplateHaskellQuotes
    * In the Template Haskell quotation 'yay'
>>> encodeword "yay"
"--- - --- ---   - ---   --- - --- ---"
>>> encodeword "im"
"- -   --- ---"
>>> encodeword "done"
"--- - -   --- --- ---   --- -   -"
>>> encodemessage "need more coffee"
"--- -   -   -   --- - -      --- ---   --- --- ---   - --- -   -      --- - --- -   --- --- ---   - - --- -   - - --- -   -   -"
>>> encodemessage "this is a morse code message"
"---   - - - -   - -   - - -      - -   - - -      - ---      --- ---   --- --- ---   - --- -   - - -   -      --- - --- -   --- --- ---   --- - - -   -      ---
 ---   -   - - -   - - -   - ---   --- --- -   -"
>>> encodemessage "to all haskell users"
"---   --- --- ---      - ---   - --- - -   - --- - -      - - - -   - ---   - - -   --- - ---   -   - --- - -   - --- - -      - - ---   - - -   -   - --- -   -
 - -"
```